TP 2 Analyse, Traitement de l'Image et Vision

Master ID3D - november 2020

Yann-Situ GAZULL
Master IF
ENS de Lyon

Matheo DUMONT
Master ID3D
Université Claude Bernard Lyon 1

Abstract

This document is a homework done by Yann-Situ Gazull and Matheo Dumont for the course ”Analyse, Traitement de l'Image et Vision” given by Saida Bouakaz.
The source code can be found on Github here https://github.com/MatheoDumont/ATIV_tps.

Index Terms

hough transform, hermit crab

Contents

I. Introduction

For this assignment, we had to implement the Hough Transform. We have implemented two primitives, the Line and the Circle, we will describe them in details further down.

II. Hough Transform

Hough Transform is a feature used to extract simple shapes from an image. From the contour of an image, we compute the parameters of the desired shape, and vote for the approximation of the parameters in a parameter space. Then, the candidates are drawn by a mean of selection.

III. Hough Transform : Lines

A. The Accumulator

To detect lines in an image of size $H * W$, we have to pick two contour and compute the equation of the line of those two contour.

The polar coordinates (ρ, θ) of a line, given two points $P_i(x_i, y_i)$ and $P_j(x_j, y_j)$ can be computed as follows :

$$||P_i - P_j|| = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$
$$\rho_{signed} = \frac{x_i y_j - x_j y_i}{||P_i - P_j||}$$

Then compute the coordinates (x_h, y_h) of the orthogonal projection of $(0, 0)$ on the line :

$$x_h = \rho_{signed} \frac{y_j - y_i}{||P_i - P_j||}$$
$$y_h = \rho_{signed} \frac{x_i - x_j}{||P_i - P_j||}$$

We can then obtain θ and ρ :

$$\theta = atan2(y_h, x_h)$$
$$\rho = |\rho_{signed}|$$

We use θ and ρ that represent our line to update the parameter space or accumulator, by rounding to the closest index. The accumulator is a matrix of size $n_\theta * n_\rho$ where we store, at $accumulator[id_\theta][id_\rho]$ the number of vote for a particular equation line, given that id_θ represent a θ and id_ρ a ρ.

As the image is in the first quarter of the 2D space, θ is between $-\pi/2$ and π and ρ is between 0 and $\sqrt{W^2 + H^2}$, where W and H are the dimension of the image. This leads to the following formulas : for a certain index id_θ and id_ρ, we get

$$\theta = \left(\frac{\pi + \frac{\pi}{2}}{n_\theta} \right) id_\theta - \frac{\pi}{2}$$
$$\rho = \frac{\sqrt{W^2 + H^2}}{n_\rho} id_\rho$$

For the accumulator, n_θ and n_ρ are arbitrarily chosen based on the precision we wants. For instance, if two parallel lines have a distance ρ between them inferior to some extent, we will choose n_ρ accordingly so those two lines are properly detected and not rounded up in a single line. If we generalise that, we will choose a n_ρ that satisfy the equation :

$$\forall d, d' \in \textbf{Lines} \,/\, d_\rho \neq d'_\rho \text{ and } d, d' \text{parallel}$$
$$\frac{\sqrt{W^2 + H^2}}{n_\rho} \leq |d_\rho - d'_\rho|$$

And the same goes for n_θ :

B. Our implementation

For our implementation, we take the threshold of an image, and pre-compute to store only the contour. So instead of iterating over all pairs of pixels of the threshold image (by finding a first pixel contour, then a second), we iterate only on our list of contour pixels. Then the complexity of it is $\mathcal{O}((n(n-1)/2)$, with n the number of pixel in the contour, instead of $\mathcal{O}((W * H)^2)$.

C. The line selection procedure

For selecting the lines, we keep those that have an accumulator value above a chosen threshold and that are accumulator local maximum (in a neighbouring that is a square of a certain length).

In order to have a threshold that is robust against scaling of images, each pair of contour point adds a vote value equal to

$$\binom{n}{2}^{-1} = \frac{2}{n(n-1)}$$

This ensure that the sum of all the final accumulator values is equal to 1, and each final accumulator value corresponds to the proportion of contour point pairs that are on the line.

D. Display the lines

In order to display the lines, we didn't use the Bresenham algorithm. Instead, we use a brute force algorithm that permits to choose the thickness of the displayed lines :

Algorithm 1 $display\_line(L, t)$

```
Require: $L$, the set of detected lines, $t$ the thickness.
Ensure: output is the matrix with the lines drawn
  for $l \in L$ do
    $\theta, \rho \leftarrow l$
    $H \leftarrow (\rho \cos\theta, \rho \sin\theta)$
    $tan_{dir}$ is the tangent of the line direction :
    $tan_{dir} \leftarrow tan(\theta + \pi/2)$
    $invtan_{dir} \leftarrow 1/tan_{dir}$
    for $P$ in $[0, cols] \times [0, rows]$ do
      $U \leftarrow P - H$
      if $|U_y - tan_{dir} * U_x| < t$ or $|U_x - invtan_{dir} * U_y| < t$ then
        $output[P_x, P_y] \leftarrow 1.$
      end if
    end for
  end for
  return $output$ of the size of the original image, with lines.
```

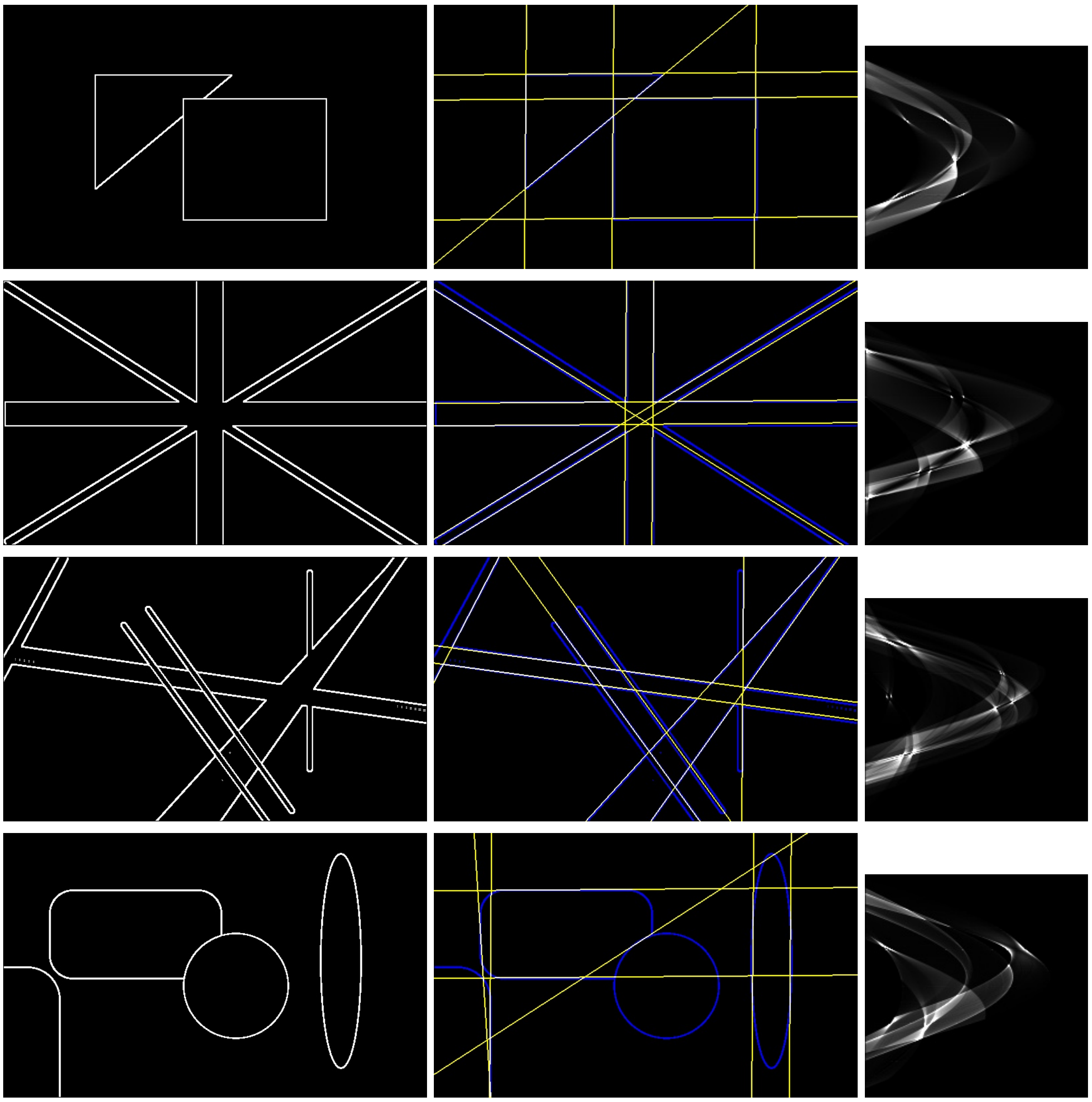

Fig. 1: Contours, line detection and accumulators of four different images.
On the second column, the selected lines are in yellow and in white when they cross contour pixels (which are in blue).
On the third column, the accumulator is represented with absciss ρ and ordinate θ, where the up-left corner is $(\theta = -\pi/2, \rho = 0)$ and the down-right is $(\theta = \pi, \rho = \rho_{max})$. The more a pixel is white, the more its accumulator value is. We can visually see parallel lines and the densest area of potential lines.

IV. Hough Transform : Circles

A. The Accumulator

For circle detection in an image, the accumulator gains an additional dimension. Indeed, our circle accumulator has 3 dimension : two for the coordinates of the center of the circles and one for its radius.

As for lines, the user can manually chose the precision of the detector by selecting the values n_x, n_y and n_r. The accumulator has a size of $n_x * n_y * n_r$ and the position (id_x, id_y, id_r) represent the circle which parameters are :

$$x = \frac{x_{max} - x_{min}}{n_x} id_x + x_{min}$$
$$y = \frac{y_{max} - y_{min}}{n_y} id_y + y_{min}$$
$$r = \frac{r_{max} - r_{min}}{n_r} id_r + r_{min}$$

For tests, we typically chose

$$\begin{array}{lll} x_{min} = 0 & x_{max} = W & n_x = W \\ y_{min} = 0 & y_{max} = H & n_y = H \\ r_{min} = 10 & r_{max} = \dfrac{W+H}{4} & n_r = r_{max} - r_{min} \end{array}$$

in order to detect circles which centers are within the image, with a radius not too small and not too big.

To detect a circle, we use a triplet of points from the contour and compute the center of the circle circumscribed to the triangle formed by the triplet.

This way, we obtain the center C of the circle and its radius is $length(C, P)$ with P one of the point of the triplet, and we just have to update the accumulator by taking care to respect the bounds min and max.

B. Our implementation

To compute the the Hough Transform of circles on an image, we iterate over all the pixels of the contour (the same way we did with the lines) and for each triplet of pixel, we compute the center of the circle circumscribed to the triangle and finally update the accumulator with the parameters. The complexity is $\mathcal{O}(n(n-1)(n-2)/6)$ where n is the number of pixels on the contour.

C. The circle selection procedure

As for lines, we keep the circles that have an accumulator value above a chosen threshold and that are accumulator local maximum (in a neighbouring that is a cube of a certain length).

As for lines, each triplet of contour points adds a vote value equal to

$$\binom{n}{3}^{-1} = \frac{6}{n(n-1)(n-2)}$$

This ensure that the sum of all the final accumulator values is equal to 1, and each final accumulator value corresponds to the proportion of contour point triplets that are on the line.

A further idea for votes is to divide the final accumulator score of each circle by its perimeter ($2\pi r$), in order to boost and detect the circles that have an important amount of vote with respect to their size (in our model, big circles are favoured).

Algorithm 2 $display\_cercle(C)$

```
Require: C, the set of detected circles.
Ensure: output is the matrix with the circles drawn
  for $c \in C$ do
    $x, y, radius \leftarrow c$
    $step \leftarrow 1/radius$
    border is the point on the perimeter of the circle
    $border \leftarrow (radius, 0.)$
    $angle \leftarrow 0.$
    while $angle < 2\pi$ do
      $x \leftarrow x + border.x$
      $y \leftarrow y + border.y$
      $output[x, y] \leftarrow 1.$
      $rotation2D(border, step)$
      $angle += step$
    end while
  end for
  return $output$ with the size of the original image, filled with circles.
```

D. Display the circles

To display the circle, we used this algorithm :

The step angle is set to $1/r$, because this ensures that we do not miss pixels when drawing the circle. Indeed, if $step = 1/radius$ we have :

$$|\cos(angle) - \cos(angle + step)| \leq step = \frac{1}{radius}$$
$$|\sin(angle) - \sin(angle + step)| \leq step = \frac{1}{radius}$$

because cos and sin are 1-Lipshitz. So we have :

$$|radius \cos(angle) - radius \cos(angle + step)| \leq 1$$
$$|radius \sin(angle) - radius \sin(angle + step)| \leq 1$$

Which means that the x and y distance between two consecutive iterations is below a pixel.

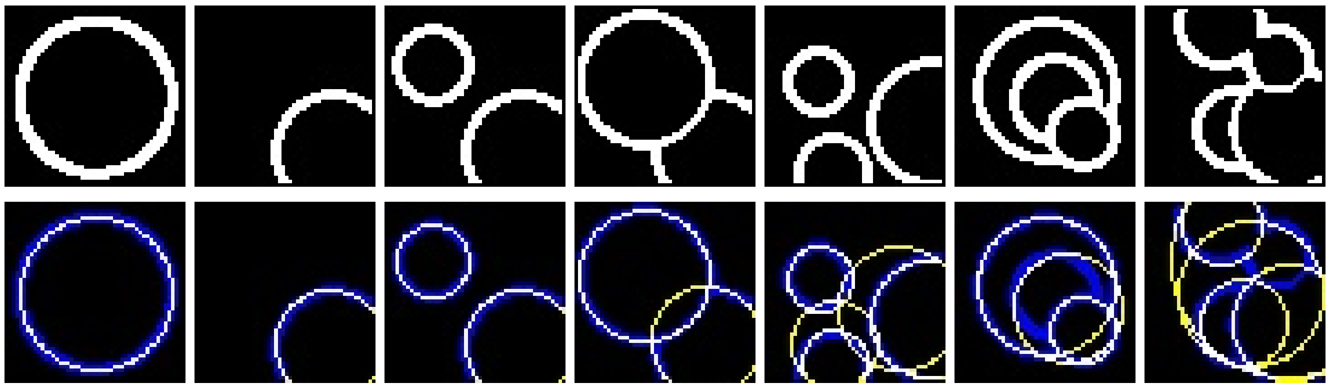

Fig. 2: Contours and circle detection on seven different images of circles.
We can see that there are some errors when multiple circles are close (on the three last images).
Moreover, as explained in *the circle selection procedure*, big circles are preferred, this can implies errors as in the last image.